Gauging the Quality of Examples for Teaching Design Patterns

Shane Sendall

Swiss Federal Institute of Technology in Lausanne (EPFL)
Software Engineering Laboratory
1015 Lausanne EPFL, Switzerland
Email: Shane.Sendall@epfl.ch

What constitutes a good example in the context of teaching design patterns? How can instructors judge whether one example is better than another one for teaching a design pattern to students? How might an example be presented to best connect with students, equipping them with the necessary know-how? In this paper, I address these questions by offering a means to judge the quality of examples for teaching design patterns: a list of desirable characteristics. The proposed list can be used to gauge the quality of examples and/or it can be used to guide the development of examples.

1. Introduction

Teaching by example gives a concrete, but natural appeal to the learning process. Einstein made the observation that “example isn't another way to teach, it is the only way to teach”. This observation is equally pertinent to the instruction of object-oriented (OO) software design.

Teaching design patterns is a challenging task. Not only is it necessary to communicate the structure of the solution offered by a design pattern, it is essential that the student understands when and where a pattern could be applied. Shalloway and Trott also raise this issue: "When you learn patterns by focusing on the solutions they present, it makes it hard to determine the situations in which a pattern applies. This only tells us what to do but not when to use it or why to do it." [ST01].

The Gang of Four were well aware of this issue when they wrote their seminal book on design patterns [GoF94]; they made the point that a pattern is not just a template to be applied on a design—it has a much wider scope of advice—it encapsulates design experience. Moreover, a good design pattern has a generative “nature”, not only showing us the characteristics of good systems, but also teaching us how to build good systems. The Gang of Four propose four defining elements of a design pattern: name, problem and its context, solution, and consequences. These four elements explain not only the “how” and the “what”, but also the “why”, “where” and “when”.

It is a “design experience” perspective, as opposed to a “design template” perspective, that needs to be understood by instructors of design patterns. It is critical that examples are chosen for their ability to uncover and express the experience encapsulated by the pattern. This means that the choice of examples and the manner in which they are presented plays a critical role in the success of teaching design patterns.

In this paper, I propose a list of desirable characteristics for examples used in the instruction of design patterns. This list can be used as a means to gauge the quality of examples and as a means to guide the development of examples.

The paper is structured in the following way: in section 2, I propose a checklist that can be used to guide the development and the choice of examples for teaching a design pattern. In section 3, I present a typical example that is commonly used to teach the Visitor pattern; also, I point out some of the example's qualities, taking into account the criteria identified in section 2. Finally, in section 4, I summarize the paper and draw conclusions.

2. Teaching Design Patterns by Example

The job of the teaching design patterns can be greatly enhanced if good examples are used. In this section, I propose a list of desirable characteristics for examples used in the instruction of design patterns. This list can be used as a means to gauge the quality of examples and as a means to guide their development. The list is given below, with a small commentary given for each characteristic:

1. The example(s) is concisely scoped to the context and problem frame of the design pattern. In other words, the example does not go beyond the scope of the design pattern's problem frame.
   Examples that exhibit this characteristic would facilitate a clear explanation of the context and problem. This would allow students to understand where the pattern could be applied.
2. The example(s) clearly illustrates the different forces that are at play in the problem and that are addressed by the pattern.
   Examples that exhibit this characteristic would facilitate a clear explanation of these forces and make it possible to point out the consequences of applying the pattern(s), both positive and negative. This would allow students to understand under what circumstances it would be appropriate to apply the pattern — providing the "when" aspect of a design pattern.
3. The example(s) lends itself to a solution, proposed by the design pattern, that is not obvious given the problem and the context, where the pattern-based solution reveals the aesthetic and utility appeal of the pattern.
   Examples that exhibit this characteristic would facilitate teaching the generative nature of the design pattern and teaching the advantages of the pattern's application in terms of maintainability and modularity. This would allow students to understand why the pattern could (or should) be applied.
4. The example requires only a minimal amount of "domain specific" information to be understood by the student(s); preferably the problem domain of the example(s) is familiar to or easily understood by the student(s).
   Examples that exhibit this characteristic would ensure that students do not have to absorb more material than needed to understand the design pattern.

An observation regarding characteristic 2 is that quite some effort may be required to demonstrate each consequence of a pattern. Furthermore, it is necessary to ensure that the problem being addressed by the examples exhibit all the forces that interact with the pattern to form the consequences of the pattern's application.

An observation regarding characteristic 4 is that it is usually desirable when teaching a design pattern to stick with an example or set of examples from a single problem domain, because the time and effort required to introduce another problem domain is usually not worth the gain in diversity.

The proposed list is clearly not specific to any pattern. Thus, the utility of this list depends on the evaluator's understanding of the pattern in focus. As a consequence, the more familiar

the evaluator is with the pattern, the more comprehensive the evaluation can be, where by familiarity with the pattern, I mean that the evaluator has a good understanding of the pattern's context, the problem that it addresses, the consequences of its use, the forces at work in the problem frame, the solution the pattern proposes, and the relationships that it has with other patterns.

In defining the list, I chose to describe the generic (desirable) characteristics in a pattern-independent way, i.e., the list is not specific to any particular design pattern. In that way, it could be used as a framework for fleshing out a specific list for each pattern, i.e., this list could be used as a guide to defining a specific list for a particular pattern.

3. An Example

In this section, I use the list that is proposed in section 2 as a guide for evaluating a concrete example about a compiler. This analysis of the example that follows is used to demonstrate how the list can be used to gauge the quality of an example, and how it can be used to point out possible areas where the example can be developed further.

The example that I use in this section is taken from the Gang of Four book [pp. 331-332, GoF94]. It is important to note that the example was used in the book to motivate the use of the Visitor pattern. As a consequence, the authors did not develop the example in a complete manner, and thus, one cannot expect it to wholly exhibit all the characteristics of the list.

The example is based on a compiler that represents programs as abstract syntax trees. Each semantic analysis phase of the compiler traverses the tree, performing specific computation at each node, in order to complete its task. Each semantic analysis phase can be applied independently of others and they may even be defined at different stages of the compiler development. I will now show how the example measures up against the list of desirable characteristics.

Analysis of the Compiler Example

The compiler example satisfactorily exhibits characteristic 1: the problem statement of the compiler example corresponds well to the context and problem scope of the pattern. Moreover, the Visitor pattern facilitates the adding of responsibilities to a hierarchy of classes in an independent and detached way, which matches nicely with the idea of additional semantic analysis phases. Thus, the example provides a nice illustration of where the pattern might be applied.

Moving onto characteristic 2, it is apparent from the description that the example does not address all consequences of the pattern; some of the missing ones include:

- Adding or removing node classes can break the visitor (e.g., cycles may be introduced that lead to infinite loops).
- Visitors will often access information of the visited nodes, forcing private information to become public, and therefore breaking encapsulation of the AST nodes.
- The traversal strategy for visitors is either placed in the nodes or entangled in the visitor code, which makes it difficult to change traversal control [Vis01].

It would be necessary to develop the example further to include the missing consequences for students to have a more complete understanding of when it would be appropriate to apply the Visitor pattern

Measuring the example against characteristic 3, it is clear that the solution proposed by the Visitor pattern has an atheistic and utility appeal to it. With the know-how of each phase encapsulated by a single object, the modularity and extensibility of the design that can be gained with the Visitor pattern is showcased. The fact that the object structure being traversed by the visitor is an abstract syntax tree also makes it more convincing. This is because very little responsibilities are placed in the nodes of an abstract syntax tree, and thus, changes are more likely to come in the form of new visitor classes, as opposed to modified or new node classes [Met02]. In this way, the example is more convincing for the application of the Visitor pattern, than say an example where the object structure is often changed (which is in general badly handled by visitors). Thus, the example illustrates nicely why the pattern might be applied.

Finally, characteristic 4 is clearly dependent on the kind of students that are targeted, and thus it is difficult to make any firm conclusions considering that I did not specify the audience's background. Nevertheless, the compiler example is sufficiently simple to understand and should not pose major problems to most software developers or future software developers.

4. Conclusion

In this paper, I proposed a list of desirable characteristics for examples used in the instruction of design patterns. The list not only covers the necessity of describing the solution proposed by the pattern and how it is applied, but also where the pattern could be applied, under what circumstances it would be appropriate to apply the pattern, and why the pattern could (or should) be applied. The list can be used as a means to gauge the quality of examples and as a means to guide their development. In defining the list, I chose to describe the generic (desirable) characteristics in a pattern-independent way, i.e., the list is not specific to any particular design pattern. In that way, it could be used as a framework for fleshing out a specific list for each pattern, i.e., this list could be used as a guide to defining a specific list for a particular pattern.

Furthermore, I used a compiler example as a subject to demonstrate how the list can be used to gauge the quality of an example, and how it can be used to point out possible areas where the example can be developed further.

The proposed list was developed out of a need to find and develop good examples for teaching design patterns. One of the main goals of this paper was to take a first step towards compiling a set of (generic) desirable characteristics for examples used in teaching design patterns. It is a starting point from which the characteristics can be refined and extended as more teaching experiences are taken into account.

References

[GoF94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides (The Gang of Four); "Design Patterns: Elements of Reusable Object-Oriented Software". Addison Wesley, 1994.

[Met02] S. Metsker; "Design Patterns Java Workbook". Addison Wesley, 2002.

[ST01] A. Shalloway, and J. Trott; "Design Patterns Explained: A New Perspective on Object-Oriented Design". Addison Wesley, 2001.

[Vis01] J. Visser; "Visitor Combination and Traversal Control". Proceedings of the 16th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '01), Tampa, USA, 2001.